TODO: Complete this page (TBS)

Overview

Resources allow users represent and reason about a wide range of resource types, including reusables(eg. a machine) and reservoirs(eg. a battery). Individual tokens (activities) can produce or consume resources (or both). Resources can have upper and lower limits, as well as bounds on instantaneous production/consumption and cumulative production/consumption.

Reasoning about resources involves two pieces:

1. **Profile:** The Profile computes the changes to the resource over time. Each production or consumption of the resource (called a transaction) affects the profile. Profile information includes the maximum and minimum possible values for the resource, and information about max/min production and consumption (both instantaneous and cumulative).
2. **Flaw and Violation Detector (FVDetector):** An FVDetector decides how to report flaws and violations given the resource profiles. For example, if one of the profile levels dips below the resources lower limit, is this a flaw, a violation, or neither?

There are a variety of Profile and FVDetector implementations available, and it is straightforward to implement your own.

To include resources in your model usually involves these steps:

1. In NDDL, extend existing resource classes to get desired behavior.
2. In NDDL, specify what type of profile is used to represent maximum/minimum values over time (default is IncrementalFlowProfile).
3. In NDDL, specify what type of detector is used to report flaws and violation (default is !ClosedWorldFVDetector).
4. In the planner configuration file specify how resource threats and unbound variables should be decided. The default file (PLASMA/trunk/config/PlannerConfig.xml) is reasonable for most cases.

For example, here is a user-defined resource that extends the Reservoir resources and uses the grounded version of the Profile and FVDetector:

```
class Battery extends Reservoir {
        string profileType;
        string detectorType;

        Battery()
        {
                super(0.0 ,0.0, 10.0); // initial, lower_limit, upper_limit
                profileType = "GroundedProfile";
                detectorType = "GroundedFVDetector";
        }
}
```

The following snippet configures the threat manager to ignore threats on the above Battery resource. Without this line, the built-in solver gets stuck because there is no built-in way to solve these threats. However, the 'Reservoir' filter in the default PlannerConfig.xml would also work:

```
<ThreatManager defaultPriority="0">
  <FlawHandler component="StandardThreatHandler"/>
  <FlawFilter class-match="Battery"/>
</ThreatManager>
```

Finally, the resource lines in the following configuration snippet are not required, but are usually what you want (and is included in the default PlannerConfig.xml):

```
<UnboundVariableManager defaultPriority="0">
  <FlawFilter var-match="start"/>
  <FlawFilter var-match="end"/>
  <FlawFilter var-match="duration"/>
  <FlawFilter class-match="Resource" var-match="time"/>
  <FlawFilter class-match="Resource" var-match="quantity"/>
  <FlawFilter class-match="Reservoir" var-match="time"/>
  <FlawFilter class-match="Reservoir" var-match="quantity"/>
  <FlawFilter class-match="Reusable" var-match="quantity"/>
  <FlawFilter component="InfiniteDynamicFilter"/>
  <FlawHandler component="StandardVariableHandler"/>
</UnboundVariableManager>
```

NEW: Resources as Constraints

This pages focuses on the traditional way to represent resources in EUROPA, which involved a token for each usage of a resource. EUROPA now includes functionality to represent directly with a constraint, without the need for an additional token. This reduces a lot of overhead computations that both propagate changes through tokens, and also reduces the number of decisions that need to be made, since there are no tokens to activate, etc.

TODO: Javier, explain how this works and the pros/cons involved. Provide example.

Common Problems

- **The solver fails to find solutions to a simple problem:** Be sure the configuration file includes resource filters as described above.
- **Profile computations are too slow:** Will GroundedProfile? work for you?

Options

The following Profile variants are available in EUROPA:

- **IncrementalFlowProfile:** Computes the tightest possible upper and lower resource bounds. Note that this should not be used if there are limits on instantaneous/cumulative production/consumption as the relevant numbers are not computed by this profile. See
- **FlowProfile** (deprecated): An early implementation of IncrementalFlowProfile that is not as efficient, but should produce identical results.
- **TimetableProfile**: A fast profile computation that results in very loose upper and lower resource bounds. The upper bounds are computed by assuming all production occurs as early as possible and all consumption occurs as late as possible, and vice versa for the lower bounds. No temporal constraints are considered when these bounds are considered. In many domains, this will profile will result in phantom flaws (flaws due to the loose bounds that cannot be resolved without specifying start/end times).

- **GroundedProfile**: A fast profile computation that can be combined with GroundedFVDetector to only report flaws or violations in the early-start schedule. The profile is computed assuming all production and consumption occur as early as possible.

The following FVDetector variants are available in EUROPA:

Combinations to Use and Avoid

Implementation Matrices

There are many possible pieces of data that can be computed by profiles and monitored by flaw/violation detectors. Here we show which ones are computed and monitored by the various profiles and detectors:

	TimetableProfile	GroundedProfile	FlowProfile	IncrementalFlowProfile
LowerLevelMin	Y	Y		
LowerLevelMax	Y	*(1)		
UpperLevelMin	Y	*(1)		
UpperLevelMax	Y	Y		
InstConsumptionMin				
InstConsumptionMax				
InstProductionMin				
InstProductionMax				
CumConsumptionMin				
CumConsumptionMax				
CumProductionMin				
CumProductionMax				

Possible New Features

Eventually, we hope to incorporate the following improvements (and bug fixes) into a future version of the Resources module:

- Non-constant upper/lower limits. For example, consider a pool of available cars that might get smaller (cars break) or larger (new cars bought) over time. The only way to represent this currently is with 'dummy' production/consumption events.
- Preferred value version of grounded profiles, so a preferred value (instead of the earliest value) could be used for grounding.
- A state resource, both for unary states (eg: on/off) and multi-state (eg: red/yellow/green). If you need a state resource immediately, ask about the hack that the DynamicEUROPA team uses.
- The GroundedProfile? does not treat instantaneous/cumulative production/consumption as 'grounded' but should.
- Re-architect flaw/violation detection so a user can pick and choose. For example, the closed-world

assumption might be desired for violations, but not for flaws.
- The OpenWorldFVDetector treats flaws in a way that is not really related to the 'open-world' concept (it doesn't report flaws due to quantity flexibility). This behavior should be separated out; not necessary as part of open-world approach, and available in closed-world approach.

If you have a need for one of these listed features, please contact the EUROPA development team and we will attempt to fast-track support for that features.c